Subject: Vectorization question
Posted by Liam E. Gumley on Mon, 11 Sep 2000 21:51:01 GMT
View Forum Message <> Reply to Message

Given the following arrays

```
a = intarr(10)
x = [2, 2, 2, 3, 3, 4]
b = [1, 3, 4, 2, 1, 8]
```

How would I vectorize the following operation

```
for i = 0, n_elements(x) - 1 do a[x[i]] = a[x[i]] + b[i]
```

To achieve this result

```
print, a, format='(10i4)'
   0   0   8   3   8   0   0   0   0   0
```

In the real-world case where this occurs, I need to repeat this kind of operation several hundred times, where the size of 'a' is around 1,000,000 and the size of 'x' is around 100,000 ('a' and 'b' are float type in the real-world case).

Many thanks for any suggestions.

Cheers,
Liam.
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: Vectorization question
Posted by Liam E. Gumley on Thu, 14 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

This will be my last post on this topic: please accept my apologies.

Liam Gumley <Liam.Gumley@ssec.wisc.edu> wrote in message news:8prqam$e4o$1@news.doit.wisc.edu...
> I forgot FORTRAN uses 1-based indices by default. What I *meant* to say was:

```
>
>       subroutine vecadd1(a, na, x, nx, b)
>       integer*4 na, nx
>       real*4 a(0:na-1), b(0:nx-1)
>       integer*4 x(0:nx-1), i
>       do i = 0, nx - 1
>         a(x(i)) = a(x(i)) + b(i)
>         end do
>         end
```

The SGI compiler doesn't like this code for some reason. So I switched back
to the original FORTRAN source, and changed the IDL wrapper function to
read:

```
x = ((long(index) > 0L) < (n_elements(a) - 1L)) + 1L
```

which converts the zero-based IDL indices to one-based FORTRAN indices. Then
everything works as advertised. This is a better approach anyway, because it
allows existing FORTRAN code to be used without modification.

Cheers,
Liam.

Subject: Re: Vectorization question
Posted by Liam E. Gumley on Thu, 14 Sep 2000 07:00:00 GMT

View Forum Message <> Reply to Message

Liam E. Gumley <Liam.Gumley@ssec.wisc.edu> wrote in message
news:39C1179A.EF9CA04E@ssec.wisc.edu...

```
>  c ...  This is the routine which does the work.
>  c ...  The arguments are defined exactly the same as in the
>  c ...  call_external procedure call in IDL.
>        subroutine vecadd1(a, na, x, nx, b)
>        integer*4 na, nx
>        real*4 a(na), b(nx)
>        integer*4 x(nx), i
>        do i = 1, nx
>          a(x(i)) = a(x(i)) + b(i)
>        end do
>        end
```

I forgot FORTRAN uses 1-based indices by default. What I *meant* to say was:

```
      subroutine vecadd1(a, na, x, nx, b)
      integer*4 na, nx
      real*4 a(0:na-1), b(0:nx-1)
      integer*4 x(0:nx-1), i
      do i = 0, nx - 1
        a(x(i)) = a(x(i)) + b(i)
      end do
      end
```

Cheers,
Liam.

Subject: Re: Vectorization question
Posted by Liam E. Gumley on Thu, 14 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"Liam E. Gumley" wrote:
> I run IDL 5.3 on SGI IRIX 6.4, so the compile went as follows:
>
> % f77 -n32 -KPIC -u -fullwarn -c vecadd.f
> % ld -n32 -o vecadd.so vecadd.o

For compiler flags on other UNIX platforms, see
$IDL_DIR/external/call_external/Fortran/Makefile

Cheers,
Liam.

Subject: Re: Vectorization question
Posted by Liam E. Gumley on Thu, 14 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"Liam E. Gumley" wrote:
> I run IDL 5.3 on SGI IRIX 6.4, so the compile went as follows:
>
> % f77 -n32 -KPIC -u -fullwarn -c vecadd.f
> % ld -n32 -o vecadd.so vecadd.o

What I meant to say was

% ld -n32 -shared -o vecadd.so vecadd.o

Cheers,
Liam.

Subject: Re: Vectorization question
Posted by Liam E. Gumley on Thu, 14 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"Liam E. Gumley" wrote:
> Given the following arrays
>
> a = intarr(10)
> x = [2, 2, 2, 3, 3, 4]
> b = [1, 3, 4, 2, 1, 8]
>
> How would I vectorize the following operation
>

> ```
> for i = 0, n_elements(x) - 1 do a[x[i]] = a[x[i]] + b[i]
> ```
>
> To achieve this result
>
> ```
> print, a, format='(10i4)'
>    0   0   8   3   8   0   0   0   0   0
> ```
>
> In the real-world case where this occurs, I need to repeat this kind of operation several hundred times, where the size of 'a' is around 1,000,000 and the size of 'x' is around 100,000 ('a' and 'b' are float type in the real-world case).

It dawned on me that this is a perfect case for an external routine. Following the example in the 'External Development Guide' for calling a FORTRAN routine with a FORTRAN wrapper, I created the following source file named vecadd.f

```
c----------
c ...  This is the interface routine called by IDL
      subroutine vecadd(argc, argv)
      integer*4 argc, argv(*), j
      j = loc(argc)
      call vecadd1(%val(argv(1)), %val(argv(2)), %val(argv(3)),
     &   %val(argv(4)), %val(argv(5)))
      end

c ...  This is the routine which does the work.
c ...  The arguments are defined exactly the same as in the
c ...  call_external procedure call in IDL.
      subroutine vecadd1(a, na, x, nx, b)
      integer*4 na, nx
      real*4 a(na), b(nx)
      integer*4 x(nx), i
      do i = 1, nx
        a(x(i)) = a(x(i)) + b(i)
      end do
      end
c----------
```

I run IDL 5.3 on SGI IRIX 6.4, so the compile went as follows:

```
% f77 -n32 -KPIC -u -fullwarn -c vecadd.f
% ld -n32 -o vecadd.so vecadd.o
```

The IDL wrapper for this routine is named vecadd.pro:

```
;----------
FUNCTION VECADD, ARRAY, INDEX, VALUE
```

```
;- Check arguments
if (n_elements(array) eq 0) then $
  message, 'Argument A is undefined'
if (n_elements(index) eq 0) then $
  message, 'Argument X is undefined'
if (n_elements(value) eq 0) then $
  message, 'Argument B is undefined'
if (n_elements(index) ne n_elements(value)) then $
  message, 'Arguments X abd B must have the same number of elements'

;- Create copies of the arguments with correct type
a = float(array)
x = (long(index) > 0L) < (n_elements(a) - 1L)
b = float(value)

;- Call the external routine
result = call_external('vecadd.so', 'vecadd_', $
  a, n_elements(a), x, n_elements(x), b)

;- Return result
return, a

END
;----------
```

So the operation I described is now quite simple:

```
a = fltarr(10)
x = [2, 2, 2, 3, 3, 4]
b = [1, 3, 4, 2, 1, 8]
result = vecadd(a, x, b)
help, result
RESULT          FLOAT     = Array[10]
print, result, format='(10i4)'
   0   8   3   8   0   0   0   0   0   0
```

The result is always returned as FLOAT, which is what I really wanted anyway. For the large arrays I described, VECADD is at least 10 times faster than a loop.

Thanks IDL!

Cheers,
Liam.
http://cimss.ssec.wisc.edu/~gumley
PS: Pavel, thanks for your suggestion as well.

Subject: Re: Vectorization question
Posted by Struan Gray on Fri, 15 Sep 2000 07:00:00 GMT

View Forum Message <> Reply to Message

Liam E. Gumley, Liam.Gumley@ssec.wisc.edu writes:

> How would I vectorize the following operation
>
> for i = 0, n_elements(x) - 1 do a[x[i]] = a[x[i]] + b[i]

I've often wondered if IDL could be modified internally to allow an implicit loop where the loop variable is only used for indexing arrays. I've hit several situations where I want to do this sort of thing, but don't want to write external routines and don't have the memory to use the 2D trick.

Oh well.

Struan
PS: I tried hard to use HISTOGRAM, but just ended up looping through the reverse indices array, which a priori is no faster.

Subject: Re: Vectorization question
Posted by promashkin on Sun, 17 Sep 2000 07:00:00 GMT

View Forum Message <> Reply to Message

I'll clock it on Monday, Craig. All I have at home is 5.1 on an AMD k6 200.
The time required on this one will not fit in one line without wrapping these days :-)
Cheers,
Pavel

> Pavel, can you compare? :-)

Subject: Re: Vectorization question
Posted by Craig Markwardt on Mon, 18 Sep 2000 07:00:00 GMT

View Forum Message <> Reply to Message

Pavel Romashkin <promashkin@cmdl.noaa.gov> writes:

```
> IDL> pavel, a, b, x, iter=10
>        5.6166667
> IDL> craig, a, b, x, iter=10
>        4.7333333
>
```

> Just as I expected, Craig's code is more efficient than mine (the magic

> of Histogram!). Good answer to those "when to use a loop" questions.
> IDL 5.3, PowerMac G4-400.

Woohoo! :-)

Craig

["Just as expected." Jeez... I would argue they are about equally
efficient to first order. My code is definitely sensitive to the
distribution of repeats. If there is a very wide range in the
number of repeats then my code will be impacted.]

```
--
 ------------------------------------------------------------ --------------
Craig B. Markwardt, Ph.D.         EMAIL:    craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 ------------------------------------------------------------ --------------
```

Subject: Re: Vectorization question
Posted by promashkin on Mon, 18 Sep 2000 07:00:00 GMT

View Forum Message <> Reply to Message

In fact, my first idea was to try Histogram to do this. But I could
never claim I reached perfection with it, as Craig did.
I tested the code, and here are the results (once I converted A and B to
FLOAT to clear the embarassing "floating illegal operand" that was
produced by my code when operating on LONG type arrays):

```
IDL> liam, a, b, x
% Array requires more memory than IDL can address.
% Execution halted at:  LIAM               40 untitled_1.pro
%                       $MAIN$
IDL> pavel, a, b, x, iter=10
      5.6166667
IDL> craig, a, b, x, iter=10
      4.7333333
```

Just as I expected, Craig's code is more efficient than mine (the magic
of Histogram!). Good answer to those "when to use a loop" questions.
IDL 5.3, PowerMac G4-400.
Cheers,
Pavel